Floating Point Exception in gm2ringsim

Adam Lyon 2016-09-01

Executive summary

The g-2 simulation code crashes with a floating point exception because we are not linking a service against a necessary library. We build without `NOUNDEFINED`, and therefore usually see such problems as understandable unresolved symbol exceptions at runtime. But the nature of the code in this case allows this linkage mistake to remain without an explicit runtime error, leading to undefined code and the hard to understand floating point exception. The solution is to add the necessary link instruction to the build.

Introduction

Tests and runs of `gm2ringsim` with `mdc0.fcl` are crashing on the first track of the first event for several builds

Branch	Flavor	git commit
feature/v7	OSX (d14 & d15)	adc1bc3
feature/mcProductMove	all flavors	d5098e7

The crash is a floating point exception - actually an integer divide by zero - in `G4AllocatorPool::Grow`.

The crash could be worked around by moving `gm2ringsim/actions/track/Trajectory.cc(.hh)` to another product (e.g. `gm2dataproducts`) and by running the tracking reconstruction.

We are running `art v2_02_01` and `art v2_03` (Mac d15) and `geant4 v4_10_02p02` with USolids. The crash occurs in both optimized and debug builds (though for OSX d15, the floating point exception in prof becomes a segmentation fault in debug).

Analysis

The `TrackingAction` service (in `gm2ringsim/actions/track`) selects Geant tracks for storage as custom trajectories (`Trajectory` in `gm2ringsim/actions/track/Trajectory.cc(.hh)`) for later analysis. `Trajectory` inherits from Geant's `G4VTrajectory`.

A traceback of the crash from a debugger is as follows:

Frame	Function
0	`G4AllocatorPool::Grow` (Crash here)
1	`G4AllocatorPool::Alloc`
2	`G4Allocator<Trajectory>::MallocSingle`
3	`Trajectory::operator new`
4	`gm2ringsim::TrackingAction::storeTrajectory`
5	`gm2ringsim::TrackingAction::preUserTrackingAction`
...	Earlier calls

To store a trajectory, the `gm2ringsim::TrackingAction::preUserTrackingAction` method calls another method `StoreTrajectory`. `StoreTrajectory` implements,

```
// Code will be identified by the file location(git commit hash):# of first line

// gm2ringsim/actions/track/TrackingAction_service.cc(d509):274
void gm2ringsim::TrackingAction::StoreTrajectory(const G4Track * currentTrack)
{
   G4TrackingManager* trackingManager = G4EventManager::GetEventManager()->GetTrackingManager();
   if( trackingManager->GetStoreTrajectory() ) { //Can be set either by user FCL or vis.mac
      Trajectory* newTraj = new Trajectory(currentTrack); // <--- CRASH
      trackingManager->SetTrajectory(newTraj);
   }
}
```

The integer divide by zero crash is occurring in `new Trajectory(currentTrack)`. Because `Trajectory` inherits from `G4VTrajectory`, we use the Geant memory management system.

Following several examples in the Geant documentation, we overload `Trajectory::operator new` to use a Geant memory allocator. In `Trajectory.cc`, we define the allocator as a **global** variable (outside of any class or namespace).

```
// gm2ringsim/actions/track/Trajectory.cc(d509):16
G4Allocator<Trajectory> aTrajectoryAllocator;
```

`TrackingAction` service does `#include "Trajectory.hh"`, and so the following occurs within the `TrackingAction_service` compilation unit.

First, `Trajectory.hh` picks up the global variable by doing,

```
// gm2ringsim/actions/track/Trajectory.hh(d509):90
extern G4Allocator<Trajectory> aTrajectoryAllocator;
```

Then it implements the overloaded `Trajectory::operator new` with,

```
// gm2ringsim/actions/track/Trajectory.hh(d509):92
inline void* Trajectory::operator new(size_t)
{
  void* aTrajectory;
  aTrajectory = (void*)aTrajectoryAllocator.MallocSingle();  // <--- CRASH
  return aTrajectory;
}
```

The crash occurs in `G4Allocator<Trajectory>::MallocSingle`.

The Geant source code for `G4Allocator::MallocSingle` is inlined in `source/global/management/include/G4Allocator.h`. It returns a pointer to a memory address returned by `G4AllocatorPool::Alloc`. `G4AllocatorPool` is responsible for creating the memory needed for `Trajectory` by growing its pool by calling `G4AllocatorPool::Grow`. Here is that method from the Geant source code...

```
// Geant source code v4.10.2.p02 global/management/src/G4AllocatorPool.cc:108
void G4AllocatorPool::Grow()
{
  // Allocate new chunk, organize it as a linked list of
  // elements of size 'esize'
  //
  G4PoolChunk* n = new G4PoolChunk(csize);
  n->next = chunks;
  chunks = n;
  nchunks++;
```

```
  const int nelem = csize/esize; // <---- CRASH with divide by zero
  char* start = n->mem;
  char* last = &start[(nelem-1)*esize];
  for (char* p=start; p<last; p+=esize)
  {
    reinterpret_cast<G4PoolLink*>(p)->next
      = reinterpret_cast<G4PoolLink*>(p+esize);
  }
  reinterpret_cast<G4PoolLink*>(last)->next = 0;
  head = reinterpret_cast<G4PoolLink*>(start);
}
```

The crash occurs because `esize` is zero, hence the division by zero. `esize` seems to be related to the total pool size. It is hard to imagine that `esize` could be zero. And in fact looking at the constructor it would seem to be impossible,

```
// Geant source code v4.10.2.p02 global/management/src/G4AllocatorPool.cc:44
// sz is the size of Trajectory = 48 bytes
G4AllocatorPool::G4AllocatorPool( unsigned int sz )
  : esize(sz<sizeof(G4PoolLink) ? sizeof(G4PoolLink) : sz),
    csize(sz<1024/2-16 ? 1024-16 : sz*10-16),
    chunks(0), head(0), nchunks(0)
{...}
```

But, from debugging and adding `cout` calls before the crash,

```
// gm2ringsim/actions/track/Trajectory.hh(d509):92 with debugging couts
inline void* Trajectory::operator new(size_t)
{
  std::cout << "BEFORE sizeof(Trajectory) = " << sizeof(Trajectory) << std::endl;
  std::cout << "BEFORE aTrajectoryAllocator::GetAllocatedSize = " << aTrajectoryAllocator.GetAllocatedSize() << std::endl;
  std::cout << "aTrajectoryAllocator.mem.GetNoPages = " << aTrajectoryAllocator.mem.GetNoPages() << std::endl;
  std::cout << "aTrajectoryAllocator.mem.GetPageSize = " << aTrajectoryAllocator.mem.GetPageSize() << std::endl;
  void* aTrajectory;
  aTrajectory = (void*)aTrajectoryAllocator.MallocSingle(); // <--- CRASH
```

```
  std::cout << "AFTER aTrajectoryAllocator::GetAllocatedSize = " << aTrajectoryAllocator.GetAllocatedSize() << std::endl;
  return aTrajectory;
}
```

we get the following output,

```
EFORE sizeof(Trajectory) = 48
BEFORE aTrajectoryAllocator::GetAllocatedSize = 0
aTrajectoryAllocator.mem.GetNoPages = 0
aTrajectoryAllocator.mem.GetPageSize = 0
Exception: EXC_ARITHMETIC (code=EXC_I386_DIV, subcode=0x0)
```

`mem.GetPageSize` is the `G4AllocatorPool` member datum `csize`. It is also impossible for that variable to be zero, but so it is.

Findings

After much debugging and a lunch meeting with the Art team, it was discovered that the `TrackingAction__service` is not linked against the library containing the `aTrajectoryAllocator` global variable (that library is `libgm2ringsim_actions_track.so(dylib)`). This is the cause of the floating point exception/integer divide by zero crash.

The `extern` in the `Trajectory.hh` file (included by `TrackingAction_service.hh`) tries to refer to `aTrajectory`, but cannot find it because the library with that symbol defined, `libgm2ringsim_actions_track.so(dylib)`, is not linked in.

We can confirm the libraries linked into `TrackingAction_service` by doing,

```
ldd $MRB_BUILDDIR/gm2ringsim/lib/libgm2ringsim_actions_track_TrackingAction_service.so
# On OSX use otool -L instead of ldd
```

and `libgm2ringsim_actions_track.so` will be mssing.

Because we compile/link without the `NOUNDEFINED` flag, we do not get a build error for undefined symbols. However we do get runtime unresolved symbol exceptions when we fail to link in a necessary library, and we fix those as they come up. One could imagine that this methodology is more efficient. But such exceptions do not occur with `extern`, and that is perhaps a fatal flaw in relying on runtime unresolved symbol exceptions. The configuration of `G4Allocator<Trajectory` and the objects it creates were thus filled with random values. Hence, the unlucky zero value for `esize` and `csize`.

We had not seen this problem before, likely because `Trajectory` is used by the tracking reconstruction code, which would link against `libgm2ringsim_actions_track.so(dylib)`, making `aTrajectory` have a known location. Running the simulation without tracking reconstruction exposes this linking problem.

Solution

The solution is to link the `TrackingAction_service` against the library with the objects from `Trajectory.cc`. Therefore, we add the following to the `gm2ringsim/actions/track/CMakeLists.txt`...

```
# gm2ringsim/actions/track/CMakeLists.txt(d509):1
art_make( LIB_LIBRARIES
          artg4_services_ActionHolder_service
          artg4_pluginActions_physicalVolumeStore_physicalVolumeStore_service
          ${XERCESCLIB}
      ${G4_LIB_LIST}
      SERVICE_LIBRARIES
      gm2geom_common_Gm2Constants_service
      gm2ringsim_actions_track     # <------ NEEDED TO AVOID CRASH
)
install_headers()
```

Doing so will make the `extern` line find the actual definition of the global `aTrajectoryAllocator`.

Further recommendations

I have two additional recommendations.

1. The Trajectory class and related code are in the global namespace. These classes should be in a specific namespace (e.g. `gm2sim::`)
2. We should build our code with the compiler option `NOUNDEFINED`. This will catch missing libraries in the link lists at build time and could have caught the problem above. Because missing libraries are a linker error under this option, builds would cease. Migrating to this option could be very time consuming, but may be advantageous.

Conclusions

This crash was difficult to understand and initially looked like a memory overrun. The implications of the `extern` and exposing a flaw in how we deal with missing library links were not considered until late in the meeting with the Art team.

The impact of this problem is:

- Delay of g-2 release by four days
- About 12 total hours of work from Adam Lyon, Tammy Walton (Fermilab g-2 postdoc) and Tom Stuttard (UCL g-2 student)
- 1 hour from part of *art* team: Jim Kowalkowski, Marc Paterno, Kyle Knoepfel and Chris Greene (0.5 hours of that over lunch)